Configuration of in-vehicle embedded systems under real-time constraints

Ricardo Santos Marques, Nicolas Navet, Françoise Simonot-Lion

Configuration of in-vehicle embedded systems under real-time constraints

Ricardo Santos Marques, Nicolas Navet, Françoise Simonot-Lion
LORIA - INPL
Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy - France
{santos, nnavet, simonot}@loria.fr
tel: +33 3 83 58 17 28, fax: +33 3 83 58 17 01

Abstract

In-vehicle embedded systems typically consist of a set of nodes exchanging applicative data ("signals") through a stack of communication protocols that includes a middleware layer. On each node, tasks, both applicative and middleware level, are subject to deadline constraints. Furthermore, signals must be produced sufficiently recently for being safely consumed on the receiver end (so-called "freshness constraint"). The goal of this study is to propose an approach for configuring nodes and communication protocols that takes these constraints into account. Precisely, the problem is, on the one hand, to set the characteristics of the middleware tasks and of the set of frames and, on the other hand, to find a feasible schedule on each node.

1 Introduction

Context of the study. In-vehicle embedded systems are made of a set of stations, termed Electronic Control Units (ECU), interconnected by a communication network as illustrated in figure 1. On each ECU, a set of *applicative level tasks* execute control algorithms (e.g. ESP, fuel injection, ACC, etc), most usually, in a periodic manner. Because functions and sensors are distributed, tasks need to exchange data, called *signals* (e.g. the speed of the vehicle) that are packed and sent into network frames when the producer and the consumer are not located onto the same ECU.

An efficient approach for easing the integration of software-based components, as well as their reusability and portability, is to provide a *middleware layer* that offers common services and a common interface. In practice, an in-vehicle middleware is made up of a set of existing communication protocols and car makers specific layers. The main goal of the middleware is to provide a set of communication services that allow to send and receive signals in an efficient manner (i.e. bandwidth consumption, temporal constraints). A set of communication services for automotive applications is defined by the industry consortium OSEK in its OSEK/VDX Communication layer specification [10].

One of the major services that an automotive middleware must offer is to pack the signals that are sent by applicative processes into frames, and to send the frames at the right points in time for ensuring the deadline constraint on each signal. This function, generally called *frame packing*, is performed according to an off-line generated configuration. OSEK/VDX Communication [10] specifies several frame transmission modes (periodic, direct and mixed); in this study, we will consider the basic and most important transmission scheme, which is the periodic transmission mode. Thus, the frame packing and its sending is not triggered by the production of signals, but periodically, according to predefined configuration.

Besides constraints on the signals exchanged, applicative tasks are most usually constrained by a *relative deadline*, which is the maximum time interval tolerated between the activation of an instance of the task and its end of execution. In order to ensure that all applicative tasks respect their relative deadline, appropriate scheduling parameters must be assigned to each of them. The most important of these parameters is the task's priority, which is the basis for tasking scheduling decision in the OSEK/VDX Operating System [11], a standard for event-triggered automotive applications.

Problem definition. Setting the scheduling and communication parameters on each node is not straightforward because, on the one hand, applicative level tasks and middleware share the CPU and thus interfere and, on the other hand, because the characteristics of the middleware tasks depend on the frame packing (see Section 2.3). Precisely, the characteristics of each middleware task must be known in order to assign a priority to applicative tasks. However, the characteristics of middleware tasks, especially their relative deadline and activation rate, depend on the set of frames resulting from the frame packing step. But, the frame packing algorithm must in turn consider both the characteristics of applicative and middleware tasks.

Goal of the paper. The goal of this article is to propose a pragmatic solution to overcome this dependency cycle and, thus, to allow the configuration of the embedded system. The proposed algorithm assigns timing and scheduling characteristics to the set of applicative and middleware tasks running on each ECU and to the set of frames transmitted over the network, such that, deadline constraints on tasks and signals will be met whenever possible.

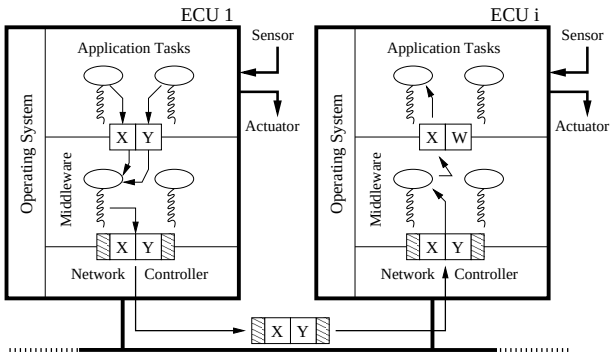

Figure 1. Part of an in-vehicle embedded system composed of two Electronic Control Units (ECUs) and a communication network. Each ECU contains a set of applicative tasks that periodically produce and consume signals through communication services provided by a middleware layer. This layer implements a mechanism that periodically packs and transmits signals into network frames.

Previous work. Under the Fixed Priority Preemptive (FPP) policy, which is the only policy available on the automotive standard OSEK/VDX Operating System (OSEK/VDX OS [11]), the Audsley algorithm [1] enables one to find the optimal priority assignment provided that all tasks are well characterized which is not the case in our context. A solution to the problem of frame configuration under schedulability constraints is implemented in the configuration tools of the middleware Volcano [2], but the algorithms of this commercial product are not published. Several heuristics are presented in [8] to build a set of frames over CAN [4] that minimizes the bandwidth consumption but without explicitly searching a feasible solution. In particular, the problem of deciding the priorities of the frames is not addressed. More recently, in [6, 13], solutions that proved to be effective on realistic experiments have been proposed but the interferences of applicative and middleware tasks are not taken into account. This study will address the latter point.

Organization of the paper. Section 2 is devoted to the presentation of the applicative tasks and signals, the middleware tasks, and the network frames. In section 3, the approach used to achieve the respect of the signals timing constraints while configuring tasks and frames is given. Section 4 introduces the several steps of the configuration algorithm, and finally, sections 5 and 6 describe the extensions proposed to existing frame packing algorithms, and the procedures allowing the configuration of the tasks.

2 System model

In the following it is assumed that, on each node, applicative level tasks and middleware tasks run on top of the OSEK/VDX OS [11], which is becoming the standard operating system for event-triggered automotive applications. Two scheduling policies are offered by OSEK/VDX OS: Fixed Priority Preemptive (FPP) and Non-Preemptive Fixed Priority (NPFP). In the remaining of this paper, only the preemptive FPP policy is assumed for the scheduling of tasks since it is more efficient in terms of schedulability but the approach can be easily extended to handle tasks scheduled under NPFP. Since applicative tasks share memory areas with middleware tasks, a resource synchronization access protocol must be used; we will assume the use of the Priority Ceiling Protocol (PCP [14]) that is provided by OSEK/VDX OS.

The communication network is considered to be a priority bus that might be CAN [4] or the J1850 [12] because at least one of these buses is today in use in almost all production cars. On such networks, each frame is assigned a priority which will serve for granting bus access according to the NPFP scheduling policy.

In this context, starting from a given set of timing characteristics of applicative tasks (e.g., periods, deadlines) and signals (e.g., freshness constraints), the goal of the configuration step is, on the one hand, to set the characteristics of the middleware tasks and the network frames, and, on the other hand, to assign a feasible priority to each applicative task. The results of this configuration step are:

- a static and unique priority assigned to each applicative task,
- the characteristics of the middleware tasks of each ECU, and
- the frame packing configuration on each ECU.

2.1 Problem input: applicative tasks and signals

The characteristics of applicative tasks and signals constitute the input data for the configuration algorithm. Each ECU i contains a finite set $\Delta_i = \{\delta_{i,1}, \delta_{i,2}, ..., \delta_{i,j}\}$ of applicative tasks where $\delta_{i,j}$ symbolizes applicative task j running on ECU i. Each $\delta_{i,j}$ is characterized by a tuple $(C_{\delta_{i,j}}, T_{\delta_{i,j}}, \bar{D}_{\delta_{i,j}}, s_{i,j}, S_{i,j})$ where:

- $C_{\delta_{i,j}}$ is the worst-case execution time for task $\delta_{i,j}$,
- $T_{\delta_{i,j}}$ is the minimum inter-arrival time of two successive instances of $\delta_{i,j}$ (termed activation period in the following),

- $\bar{D}_{\delta_{i,j}}$ is the relative deadline, that is the maximum tolerable time interval between the activation of an instance and the end of execution of this instance,
- $s_{i,j}$ is the output signal produced by task $\delta_{i,j}$ that is characterized by its size $C_{s_{i,j}}$ in bits. Its production period is equal to the period of task $\delta_{i,j}$,
- and $S_{i,j}$ is the set of signals consumed by $\delta_{i,j}$ that are characterized by pairs $(s_{i',j'}, \mathcal{F}_{\delta_{i,j}}^{s_{i',j'}})$ where
  - $s_{i',j'}$ is a signal produced in ECU i' by applicative task $\delta_{i',j'}$, and
  - $\mathcal{F}_{\delta_{i,j}}^{s_{i',j'}}$ is the freshness constraint requested by $\delta_{i,j}$ on signal $s_{i',j'}$.

This freshness constraint imposes a limit on the maximum age of the signal value. Precisely, the time interval between the consumption of a signal value and the activation of the instance of the task that produced that value, must be less than, or equal to, the freshness constraint. For instance, the number of RPM needed to compute the speed of the vehicle must have been read at the engine controller no more than 20ms before the speed is used by the active suspension. It is assumed that input data of a task is collected upon its activation since it is the first time instant where the task can begin to be executed.

Figure 2 illustrates this constraint with respect to signal $s_{i',j'}$: when an instance of applicative task $\delta_{i,j}$ consumes the value α, its age must be smaller than or equal to the freshness constraint $\mathcal{F}_{\delta_{i,j}}^{s_{i',j'}}$.

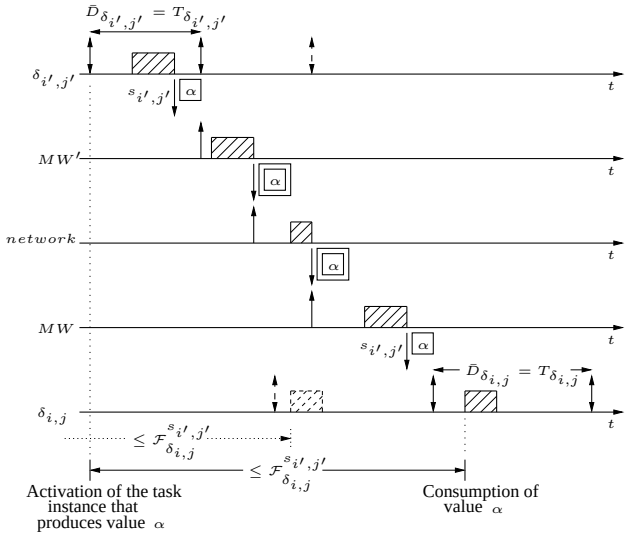

Figure 2. Scenario in which applicative task $\delta_{i,j}$ consumes signal $s_{i',j'}$ produced by applicative task $\delta_{i',j'}$. The configuration of applicative and middleware tasks and of the network frame must guarantee that the age of the values of $s_{i',j'}$ respects the freshness constraint $\mathcal{F}_{\delta_{i,j}}^{s_{i',j'}}$ associated to $s_{i',j'}$. Double square boxes represent a network frame that, in this case, carries value α.

2.2 Problem output: network frames

Starting from the signals, the frame packing step will produce a set of frames that minimizes bandwidth consumption and respects temporal constraints. Each frame $f_{i,k}$ belonging to the set of frames F_i, transmitted by ECU i, will be characterized by a tuple $(C_{f_{,i,k}}, T_{f_{i,k}}, \bar{D}_{f_{i,k}}, S_{f_{i,k}}, P_{f_{i,k}})$ where:

- $C_{f_{i,k}}$ is the size of $f_{i,k}$ in bits,
- $T_{f_{i,k}}$ is the transmission period,
- $\bar{D}_{f_{i,k}}$ is the relative deadline defining the maximum time interval tolerated between the transmission request and the reception of the frame at all nodes,
- $S_{f_{i,k}}$ is the set of signals composing frame $f_{i,k}$, and
- $P_{f_{i,k}}$ is the priority of $f_{i,k}$, unique on the whole system, for the Medium Access Control protocol.

2.3 Problem output: characteristics of middleware tasks

OSEK/VDX OS conformance requires that at least 16 tasks (or even 8 with other conformance classes) can be handled by the OS. Thus, for the sake of portability, the number of tasks, including middleware tasks, must not be greater than 16 or 8, depending on the conformance class (see [11] for more details).

Given this strong constraint, in the following, we assume that the middleware is implemented in two distinct tasks. A task is responsible for handling time-triggered events, OSEK/VDX OS alarms for periodic transmission in the following, and the other is in charge of event-triggered ones, frame arrival interrupts in the following. This way, one maximizes the number of possible applicative level tasks on each ECU.

Precisely, the first middleware task is responsible for unpacking newly arrived frames and for writing the signal values at the right memory areas. On CAN, the clocks on the different ECUs are unsynchronized and the exact time interval between two consecutive frame arrivals cannot be determined. The first task is thus considered as sporadic [5] where its activation period is equal to the time needed to transmit the smallest frame that can be received. The configuration of this task, denoted ω_i in ECU i, is given by a tuple $(C_{\omega_i}, T_{\omega_i}, \bar{D}_{\omega_i}, S_{\omega_i})$ where:

- C_{ω_i} is the time needed to execute ω_i,
- T_{ω_i} is the activation period representing the smallest time interval between two consecutive activations,
- $\bar{D}_{\omega_i}$ is the relative deadline defining the maximum time interval accepted between the activation and the end of execution of ω_i, and

- S_{ω_i} is the set of signals that ω_i updates: $S_{\omega_i} = \bigcup_j S_{i,j}$ (i.e. the set of signals consumed by ECU i).

The second middleware task is responsible for transmitting frames, and therefore, its characteristics depend on the frame packing configuration. This task can be implemented as a multiframe task introduced in [7]. This type of task is characterized by a unique activation period and a set of different execution times corresponding to successive instances of the task. In our case, this middleware task has different execution times because successive instances transmit a different set of frames. The configuration of this task in each ECU i, denoted ϕ_i, is characterized by a tuple $(C_{\phi_i}, T_{\phi_i}, \bar{D}_{\phi_i})$ where:

- $C_{\phi_i} = \{c_{\phi_i}^{(0)}, c_{\phi_i}^{(1)}, ..., c_{\phi_i}^{(n-1)}\}$ is a set of n execution times; the execution time of instance a of task ϕ_i is given by $c^{(a \bmod n)}$ where $a \geq 0$, and the execution time of subsequent instances is obtained by cycling through the set C_{ϕ_i} in order,
- T_{ϕ_i} is the activation period defining the minimum time interval between two consecutive activations, and
- $\bar{D}_{\phi_i}$ is the relative deadline identifying the maximum time interval tolerated between the activation and the end of execution of task ϕ_i.

From the frame packing configuration, one derives the execution times and the activation period of the multiframe task ϕ_i as follows. Let the set $Q_i = \{(Q_{f_{i,1}}, T_{f_{i,1}}), ..., (Q_{f_{i,k}}, T_{f_{i,k}})\}$ where $Q_{f_{i,k}}$ is the time needed to construct and request transmission of frame $f_{i,k}$, and $T_{f_{i,k}}$ is the transmission period of the frame. The activation period of ϕ_i, T_{ϕ_i}, is simply $\gcd(T_{f_{i,1}}, ..., T_{f_{i,k}})$. For each activation date during the first hyperperiod, $\text{lcm}(T_{f_{i,1}}, ..., T_{f_{i,k}})$, one determines the frames that are to be sent and, thus, the set of execution times for ϕ_i:

$$\forall\, 0 \leq a \leq \text{lcm}(T_{f_{i,1}}, ..., T_{f_{i,k}})/T_{\phi_i} - 1,$$

$$c_{\phi_i}^{(a)} = \sum_{\{k \mid a \cdot T_\phi \bmod T_{f_{i,k}} = 0\}} Q_{f_{i,k}}$$

It is assumed that these two middleware tasks have the two highest priorities. In the first place, since communication services are the most priority on each ECU, it avoids losing frames due for instance to buffer overflow. On the other hand, middleware tasks can prevent a faulty task from jeopardizing the system's behaviour, for instance, by consuming all the CPU time due to a software bug.

With OSEK/VDX OS, ISRs have necessarily a higher level of priority than tasks. In our context ω_i has a higher priority than ϕ_i. This is justified by the fact that the frame reception task ω_i would be most efficiently implemented as an OSEK/VDX OS interrupt service routine (ISR) triggered by the communication controller interrupt, while the task ϕ_i, which is activated periodically by the OSEK alarm services, would be implemented as a classical OS task.

3 Guaranteeing freshness constraints

The aim is to guarantee that any signal will meet its freshness constraints at the time when it is consumed. Since nodes are not synchronized, all consumption times are possible. One has thus to ensure that at any time, the value of a signal that is available to tasks, respects its constraints.

The maximum age that any instance of signal $s_{i',j'}$ can reach is the longest time duration between the activation of the producer task instance and the moment where the value of the signal is updated at the receiver end (availability of the next instance of the signal). Our approach aims to ensure that the maximum age of any signal available for consumption is less than, or equal to, its freshness constraints.

3.1 Characterization of the maximum age

To determine the maximum age of a signal value let us introduce:

- $A_{\delta_{i',j',n}}$, the activation instant of the n-th instance of task $\delta_{i',j'}$, and
- $R_{\omega_i}^{s_{i',j'}}$, the worst-case end-to-end response time for signal $s_{i',j'}$ that is the longest time between the instant at which middleware task ω_i writes a value of signal $s_{i',j'}$ in shared memory, and the activation instant of the instance of the applicative task that produced the value.

Figure 3 illustrates the case where the age of a signal is maximized. One denotes by $s_{i',j',n}$, the n-th produced value of signal $s_{i',j'}$. It starts to age at the n-th activation of its producer task, that is, at instant $A_{\delta_{i',j',n}}$.

Value $s_{i',j',n}$ will be updated by $s_{i',j',n+1}$, produced by the task instance activated $T_{\delta_{i',j'}}$ units of time after $A_{\delta_{i',j',n}}$. In the worst case, value $s_{i',j',n+1}$ is written by the middleware task ω_i in shared memory $R_{\omega_i}^{s_{i',j'}}$ units of time later. Value $s_{i',j',n}$ is then updated $T_{\delta_{i',j'}} + R_{\omega_i}^{s_{i',j'}}$ units of time after the activation of its producer task instance. This is therefore the maximum age that any value of signal $s_{i',j'}$ can reach.

One denotes $\bar{\mathcal{F}}^{s_{i,j}} = \min_{\forall y,z \mid s_{i,j} \in S_{y,z}} (\mathcal{F}_{\delta_{y,z}}^{s_{i,j}})$, the most stringent freshness constraint for signal $s_{i,j}$. To guarantee that every signal $s_{i,j}$ respects $\bar{\mathcal{F}}^{s_{i,j}}$, the following equation must be verified:

$$\forall\, i, j, x,\ s_{i,j} \in S_{\omega_x} \qquad T_{\delta_{i,j}} + R_{\omega_x}^{s_{i,j}} \leq \bar{\mathcal{F}}^{s_{i,j}} \qquad (1)$$

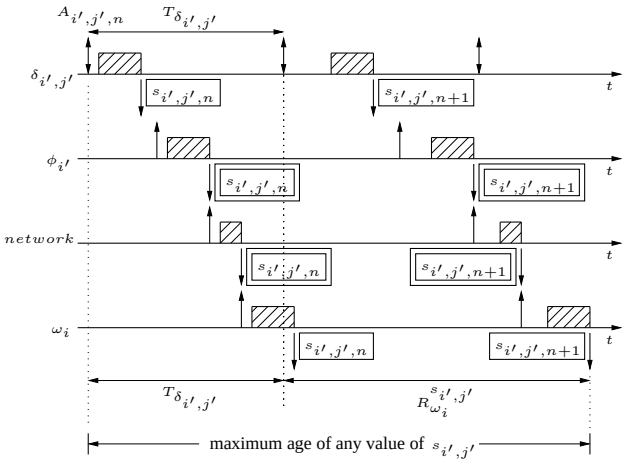

Figure 3. Illustration of the maximum age that the signal $s_{i',j',n}$ can reach. Its age begins at $A_{\delta_{i',j',n}}$, the n-th activation of its producer task. In the worst case, the next instance of the signal, $s_{i',j',n+1}$, is available $T_{\delta_{i',j'}} + R^{s_{i',j'}}_{\omega_i}$ units of time later. The maximum age that any signal $s_{i',j'}$ can reach is $T_{\delta_{i',j'}} + R^{s_{i',j'}}_{\omega_i}$. Double square boxes represent a network frame.

The period of production of the signal, $T_{\delta_{i,j}}$, is inherited from its producer applicative task. The value of the worst-case end-to-end response time $R^{s_{i,j}}_{\omega_i}$ depends on the characteristics of applicative and middleware tasks, frames and the scheduling on ECUs and network. In the rest of the paper, we propose an algorithm that will try to configure the system in such a way that equation 1 is met.

3.2 Worst-case end-to-end response time of a signal

The end-to-end response time of a signal $s_{i',j'}$, illustrated in figure 4, depends on:

1. the interval needed by the applicative task to produce the signal (response time of the task),
2. the activation time of the middleware task that will build up the frame and request its transmission. Precisely, it depends on the time between the production of the signal (end-of-execution of the producer task) and the activation of the middleware task that will actually send the frame containing the signal,
3. the response time of the middleware task sending the frame,
4. the response time of the frame containing the signal,
5. the time needed to unpack the frame at the receiver end and make the signal available to consumer tasks.

Durations 1,3,4 and 5 are mutually independent so the worst-case situation occurs with the maximum of these durations. The worst-case response time of the applicative task, $R_{\delta_{i',j'}}$, can be computed using classical FPP schedulability analysis with PCP protocol [16] (since memory areas are shared between applicative and middleware tasks) and higher priority multiframe and sporadic tasks [7, 15]. The worst-case response time of the sending middleware task, $R_{\phi_{i'}}$, is computed in a similar manner, the only higher priority workload being the middleware task that unpacks incoming frames. As explained in 2.3, this latter task can be conveniently implemented as an OSEK/VDX ISR. The worst-case response time of the frame containing the signal, $R_{f_{i',k'}}$, is determined using the NPFP response time analysis proposed by Tindell et al [17]. Finally, the worst-case time needed to unpack the frame at the receiver end, R_{ω_i}, is simply the execution time of the ISR that takes into account the longest critical section of the applicative tasks [14].

Duration 2 depends on the periods of the applicative level task, $T_{\delta_{i',j'}}$ (input data of the problem), and the frame containing the signal, $T_{f_{i',k'}}$ (results of the frame packing algorithm). The aim is to evaluate the worst-case interval between the availability of the signal and the activation of the middleware task that will send the frame. Two cases arise:

1. $T_{\delta_{i',j'}} < T_{f_{i',k'}}$
2. $T_{\delta_{i',j'}} \geq T_{f_{i',k'}}$

Case 1 implies that some produced signals will be lost and it is thus ruled out since the frame packing algorithm must avoid this. In case 2, the maximum delay cannot be larger than $T_{f_{i',k'}}$, corresponding to the scenario where the signal is available immediately after the activation of the middleware task.

The expression for the worst-case end-to-end response time of a signal $s_{i',j'}$ is thus simply: $R^{s_{i',j'}}_{\omega_i} = R_{\delta_{i',j'}} + T_{f_{i',k'}} + R_{\phi_{i'}} + R_{f_{i',k'}} + R_{\omega_i}$ where $s_{i',j'} \in S_{\omega_i}$ and $s_{i',j'} \in S_{f_{i',k'}}$.

4 Configuration algorithm

The goal of the configuration algorithm is 1) to characterize applicative and middleware tasks and frames and 2) to define the relative priorities for the scheduling on ECUs and network, in such a way that, the worst-case end-to-end response time of each signal and the deadline of each task is met.

In order to assign priorities to applicative tasks, one needs to characterize the middleware tasks to precisely quantify their interference. The parameters of these middleware tasks depend on the result of the frame packing algorithm. But, the frame packing algorithm needs the knowledge of the worst-case response times of applicative and middleware tasks in order to assign a deadline to each frame that guarantees the signal freshness constraint. To overcome this dependency cycle, we propose a three-step algorithm that is presented in figure 5.

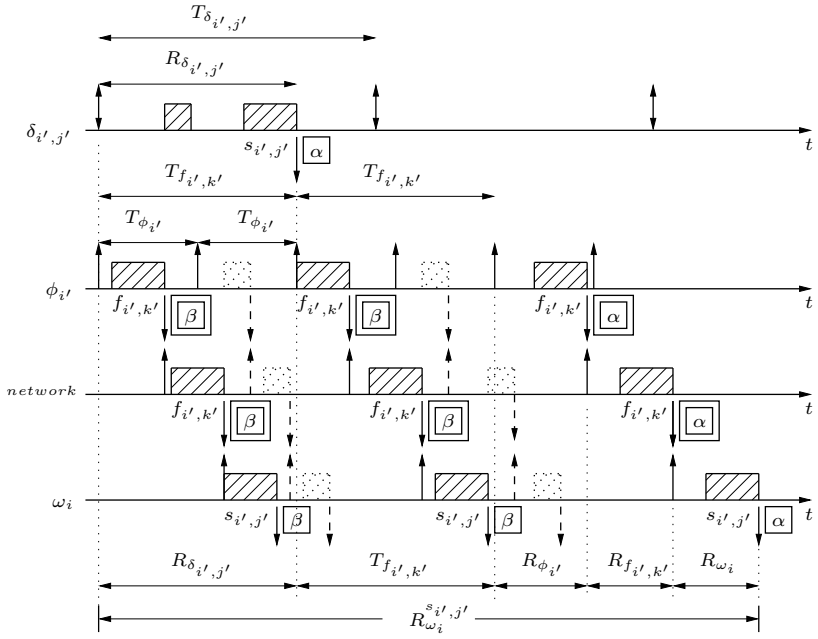

Figure 4. Scenario leading to the worst-case end-to-end response time $R_{\omega_i}^{s_{i',j'}}$ of signal $s_{i',j'}$, from the activation of its producer task $\delta_{i',j'}$ until the moment it is available for consumption in the receiver end. Shadowed boxes represent resp. the execution and the transmission of middleware tasks, resp. frames, that are not involved in the exchange of $s_{i',j'}$.

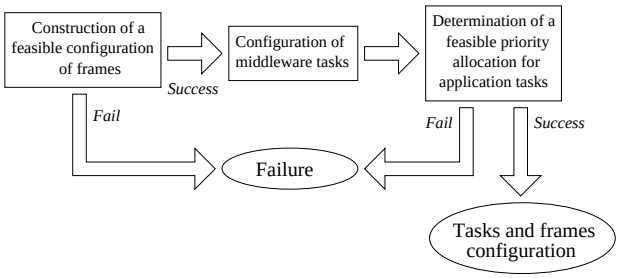

Figure 5. Steps of the algorithm for the configuration of applicative and middleware tasks, and network frames.

The first step is the execution a frame packing algorithm such as the *Bi-Directional Frequency Fit* (BDFF) from [13] or the *Bandwidth Best Fit decreasing* (BBFd) from [6]. Basically the idea is to find the frame configuration (signals composing a frame, period and deadline of the frames) that minimizes the bandwidth consumption while meeting feasibility constraints. The algorithms in [13] and [6] do not consider the delays induced by applicative and middleware tasks. To take these delays into account and break the dependency cycle, we assume that the worst-case tolerated behaviour occurs. If freshness constraints are met under these assumptions, they will be necessarily met with the actual worst-case response times. The successful output of this step is a feasible global frame configuration. When no feasible configuration is found, the freshness constraints have to be relaxed or the applicative tasks re-designed.

The second step consists of setting the parameters of the middleware tasks of each ECU starting from the frame configuration. Precisely, one has to determine the execution time and activation period.

Finally, third step tries to determine a feasible priority allocation for the set of applicative tasks of each ECU. One will make use of the Audsley priority assignment algorithm presented in [1]. It must however be taken into account that middleware tasks have the two highest priorities. If no feasible priority allocation is obtained, either the task set or the constraints have to be re-worked.

5 Extensions to existing frame packing algorithms

This section details the extensions proposed to existing frame packing algorithms like BDFF [13] or BBFd [6]. These algorithms construct the set of frames sent by each ECU from the set of signals, such that, the freshness constraints associated to signals are met and the bandwidth consumption is minimized.

Steps of the frame packing algorithms. They are composed of two distinct parts as illustrated in figure 6. The first part aims at constructing a set of frames whose characteristics minimize the bandwidth consumption. In [13] and [6], the algorithms involved have similarities with the bin-packing problem [3] but the aim is to reduce the bandwidth usage instead of the number of boxes. The second

part deals with the schedulability of the proposed solution that is tested with the Audsley algorithm[1]. If the feasibility test fails then a frame decomposition scheme tries to split frames in such a way that the deadlines of the frames are increased and some workload is moved at lower priority levels, augmenting thus the likelihood of obtaining a feasible solution.

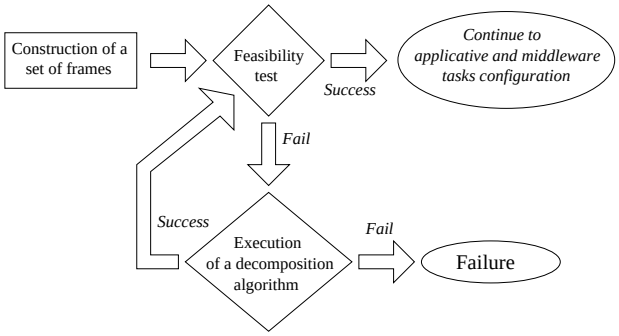

Figure 6. Description of a frame-packing strategy [13, 6]. The first step aims at constructing a solution that minimizes the bandwidth consumption. The second step deals with the feasibility of the solution. If no priority allocation exists then a frame decomposition algorithm is executed.

While the set of frames is being built, to decide if a signal can be inserted in a specific frame, the deadline of the frame with this signal is determined. However, this computation does not take into account the delays induced by applicative and middleware tasks, and the same problem exists in the decomposition scheme.

Worst-case scenarios. In the following, extensions for considering the worst-case delays induced by the tasks participating in a signal exchange are presented. The exchange of a signal involves the execution of tasks on the sender and receiver sides. Let us examine the exchange of signal $s_{i',j'}$ carried by frame $f_{i',k'}$.

- Receiver side: middleware task ω_i

The only task involved on the receiver end is the one responsible for the handling of $f_{i',k'}$, task ω_i in figure 4, which has the highest priority on each ECU. Its worst-case response time is simply its worst-case execution time plus the maximum blocking time due to shared resources, that is, as PCP is used, the longest critical section for consuming signals of the applicative tasks running in ECU i.

[1]The Audsley algorithm [1] has been shown in [13] to be also optimal for NPFP if the blocking factor (i.e. the maximum time interval during which one frame can be delayed by a lower priority frame) is equal for all frames. In our context, we assume the existence of at least one low-priority frame having the largest size allowed by the protocol. Indeed, in automotive systems, diagnostic and network management frames of arbitrary size are also exchanged in addition to the real-time traffic.

- Sender side: middleware task $\phi_{i'}$

On the sender side, middleware task $\phi_{i'}$, see figure 4, transmits $f_{i',k'}$. We assume a worst-case scenario consisting of $f_{i',k'}$ and a one-signal-per-frame packing (for all signals not in $f_{i',k'}$), and thus, as almost many transmission requests as signals. The blocking time due to shared resources is equal to the longest critical section for producing signals of the applicative tasks running in ECU i'. In addition, it suffers from the interference of the higher priority task $\omega_{i'}$, whose worst-case scenario will be discussed later.

- Sender side: applicative task $\delta_{i',j'}$

The applicative task $\delta_{i',j'}$ produces $s_{i',j'}$. Its worst-case execution scenario occurs when it runs at the lowest priority level such that $\bar{D}_{\delta_{i',j'}}$ is respected. In addition to the higher priority applicative tasks and blocking time (shared memory areas for storing produced data and reading signals for consumption), the worst-case response time must take into account the maximum interference of middleware tasks $\phi_{i'}$ and $\omega_{i'}$.

- Sender side: middleware task $\omega_{i'}$

Task $\omega_{i'}$, with the highest priority, does not participate directly in the exchange of $s_{i',j'}$ but it interferes with all tasks running on ECU i'. Its worst-case scenario is the handling of the largest frame that can be consumed in ECU i' with an activation period equal to the time needed to transmit the smallest frame that is consumed in ECU i'.

Frame deadline. From the delays corresponding to the worst-case scenario, the frame deadline can be set. The frame deadline that is needed for guaranteeing the freshness of a signal, is the largest value such that equation 1 is met (see section 3.1). Then, the deadline of the frame is simply the most stringent deadline constraint induced by the signals composing the frame.

6 Configuration of applicative and middleware tasks

This section explains how the characteristics of the middleware and applicative tasks are obtained after the frame packing configuration step.

Middleware tasks. Since the priorities of middleware tasks are already defined due to implementation and efficiency constraints (see 2.3), one has just to determine their execution time, activation period and relative deadline needed for the schedulability analysis. On each ECU i, the worst-case characteristics of task ω_i are the following:

- C_{ω_i} is the time necessary to handle the largest frame that ω_i can receive,

- T_{ω_i} is the time to transmit the smallest frame that ECU i can receive,
- $\bar{D}_{\omega_i}$ is equal to T_{ω_i}, otherwise frames can be lost.

The characteristics of task ϕ_i are :

- C_{ϕ_i} and T_{ϕ_i} are implied by the frame packing configuration as explained in section 2.3,
- $\bar{D}_{\phi_i}$ is equal to T_{ϕ_i}, otherwise a set of frames may suffer a delay caused by the sending of a previous set of frames. This is not taken into account since it corresponds to a situation where applicative tasks cannot execute and, thus, cannot produce signals to be transmitted.

Applicative tasks. A feasible priority allocation for the set of applicative tasks is searched for with the optimal Audsley algorithm [1]: if a solution exists then it will necessarily be found. The strategy is to start from the lowest priority (m) and to look for a task that is feasible at this level. In case of failure, one can conclude that the set of tasks is not schedulable. The first feasible task at level m is assigned to that priority, then the algorithm tries to find a feasible task at level $m-1$, and so on until priority 1, the highest priority of the system. Besides other higher priority tasks, the feasibility test (i.e. worst-case response time computation) used in the Audsley algorithm must consider the blocking time caused by the PCP protocol.

Conclusion

In this study, a method for the configuration of in-vehicle embedded systems is proposed. Scheduling parameters are assigned to applicative and middleware tasks and to network frames in such a way that, tasks and signals timing constraints are respected. All the technical elements used in this study (FPP and NPFP feasibility, Audsley algorithm, frame packing strategies, ...) are well known but, to our best knowledge, such a comprehensive approach is original and of practical interest.

The proposed method runs in three steps. The first is the execution of an existing frame packing algorithm that constructs a feasible and bandwidth minimizing set of frames. Some extensions are proposed to take into account the delays induced by applicative and middleware tasks. These extensions are based on worst-case scenarios, since while the frame packing is executing, the precise scheduling parameters of the tasks are not known. The second step is the configuration of middleware tasks. The final step tries to determine a feasible scheduling for applicative tasks.

We are currently working on a tool that generates the OSEK Implementation Language configuration [9] corresponding to the best solution found. It will help to automatize the design of validated automotive embedded systems.

References

[1] N. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS164, University of York, November 1991.

[2] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a revolution in on-board communications. Technical report, Volvo, 1999.

[3] E. Coffman, M. Garey, and D. Johnson. *Approximation algorithms for bin packing: a survey*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1996.

[4] ISO. *ISO 11898 - Road vehicles - Interchange of digital information - Controller Area Network for high-speed Communication*. International Standard Organization, 1994. ISO 11898.

[5] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[6] R. S. Marques, N. Navet, and F. Simonot-Lion. Frame packing under real-time constraints. In *5th IFAC International Conference on Fieldbus Systems and their Applications - FeT'2003, Aveiro, Portugal*, pages 185–192, July 2003.

[7] A. Mok and D. Chen. A multiframe model for real-time tasks. In *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96)*, page 22. IEEE Computer Society, 1996.

[8] C. Norström, K. Sandström, and M. Ahlmark. Frame packing in real-time communication. Technical report, Mälardalen Real-Time Research Center, 2000.

[9] OSEK Consortium. OIL:OSEK implementation language, version 2.5, July 2004. Available at `http://www.osek-vdx.org`.

[10] OSEK Consortium. OSEK/VDX communication specification, version 3.0.3, July 2004. Available at `http://www.osek-vdx.org`.

[11] OSEK Consortium. OSEK/VDX operating system, version 2.2.3, February 2005. Available at `http://www.osek-vdx.org`.

[12] SAE. SAE J1850 standard, class B data communications network interface, May 1994.

[13] R. Saket and N. Navet. Frame packing algorithms for automotive applications. Technical Report INRIA RR-4998, 2003.

[14] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[15] H. Takada and K. Sakamura. Schedulability of generalized multiframe task sets under static priority assignment. In *RTCSA '97: Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications (RTCSA '97)*. IEEE Computer Society, 1997.

[16] K. Tindell. An extendible approach for analyzing fixed priority hard real-time tasks. Technical Report YCS189, University of York, 1992.

[17] K. Tindell, A. Burns, and A. Wellings. Calculating Controller Area Network (CAN) message response times. *Control Eng. Practice*, 3(8):1163–1169, 1995.